Veronica Sonzini

Music Player

March 28, 2019

Product Overview

This music player was created for iOS devices, using the language Swift 4. The IDE used was XCode, which is Apple's own IDE for creating native apps.

In order to persist the data I decided to use Core Data, which is a persistence framework provided by apple.

Data Model

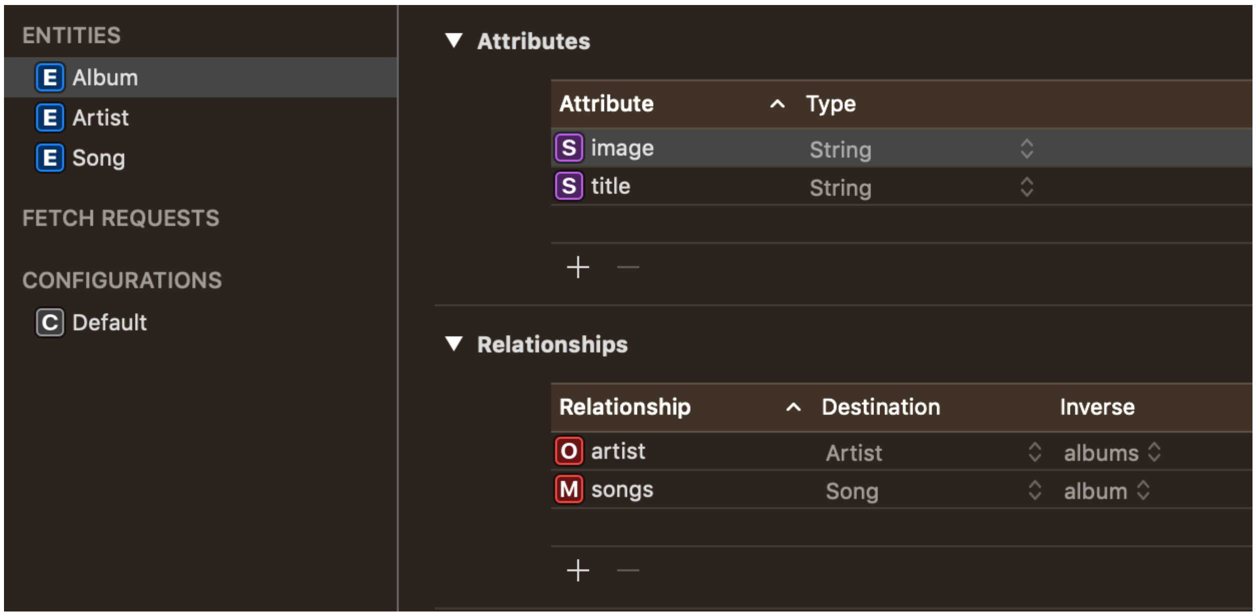
Model data using Xcode's model editor

Entity: is a class definition in Core Data. For this application, I had to create 3 different entities

- Song
- Album
- Artist
- Playlist

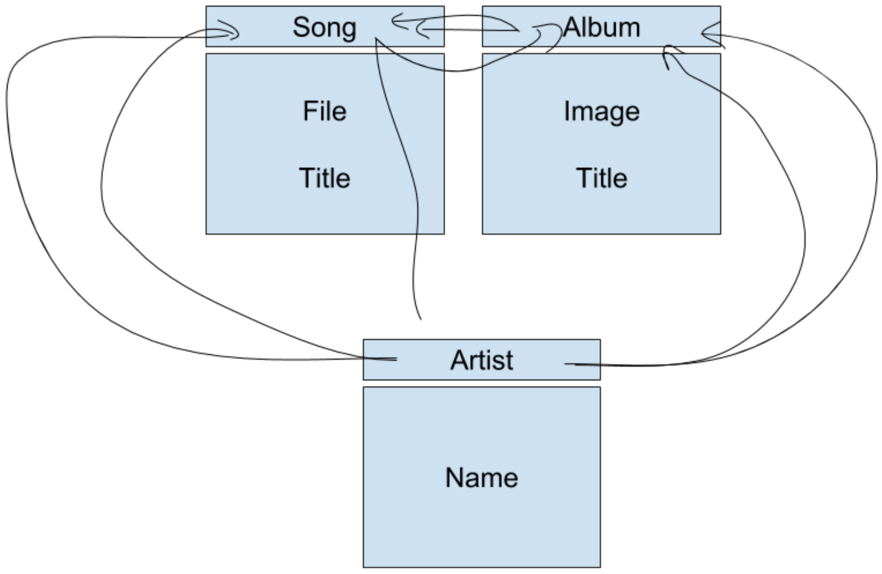
Attribute: is a piece of information attached to a particular entity. For example, in this case a **Song** entity might have a **title** attribute.

Relationship: a relationship is a link between multiple entities. In Core Data, relationships between two entities are called to-one relationships, while those between one and many entities are called to-many relationships.



Attribute	Type
S image	String
S title	String

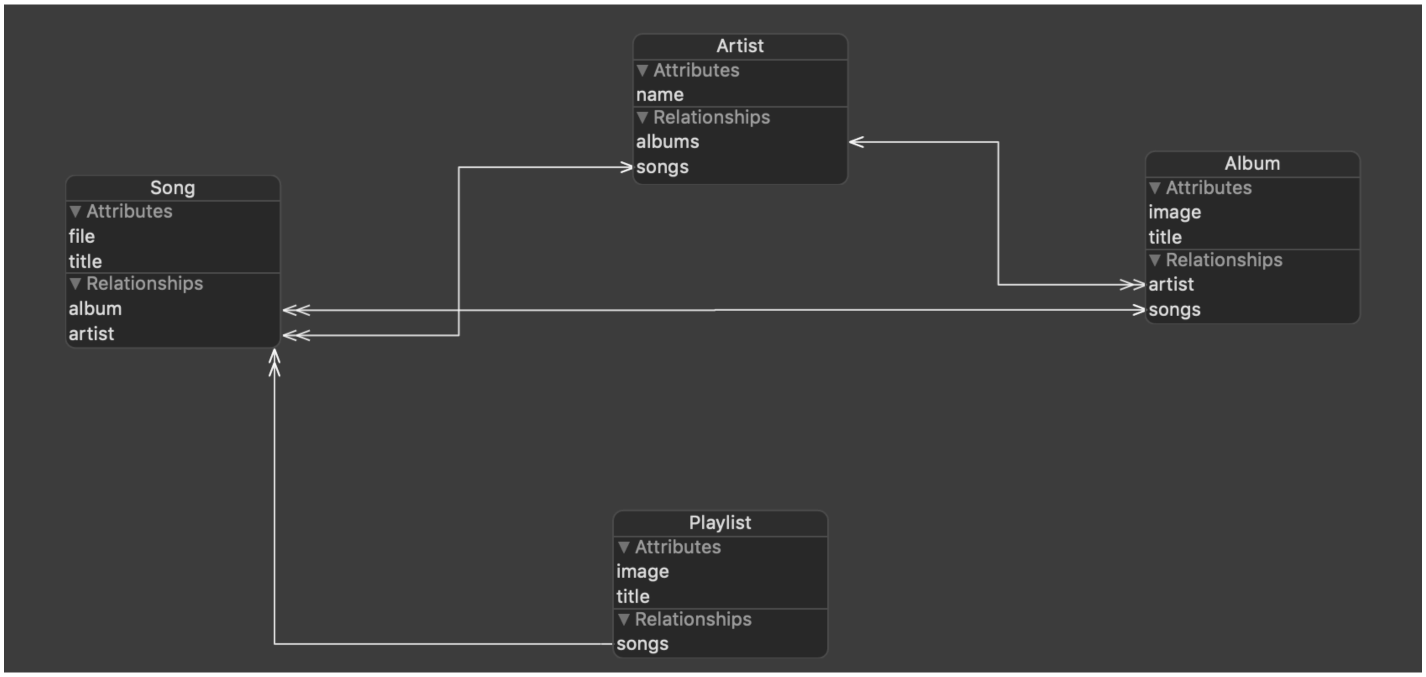
Relationship	Destination	Inverse
O artist	Artist	albums
M songs	Song	album



Here we can see the relationships I created between the entities:

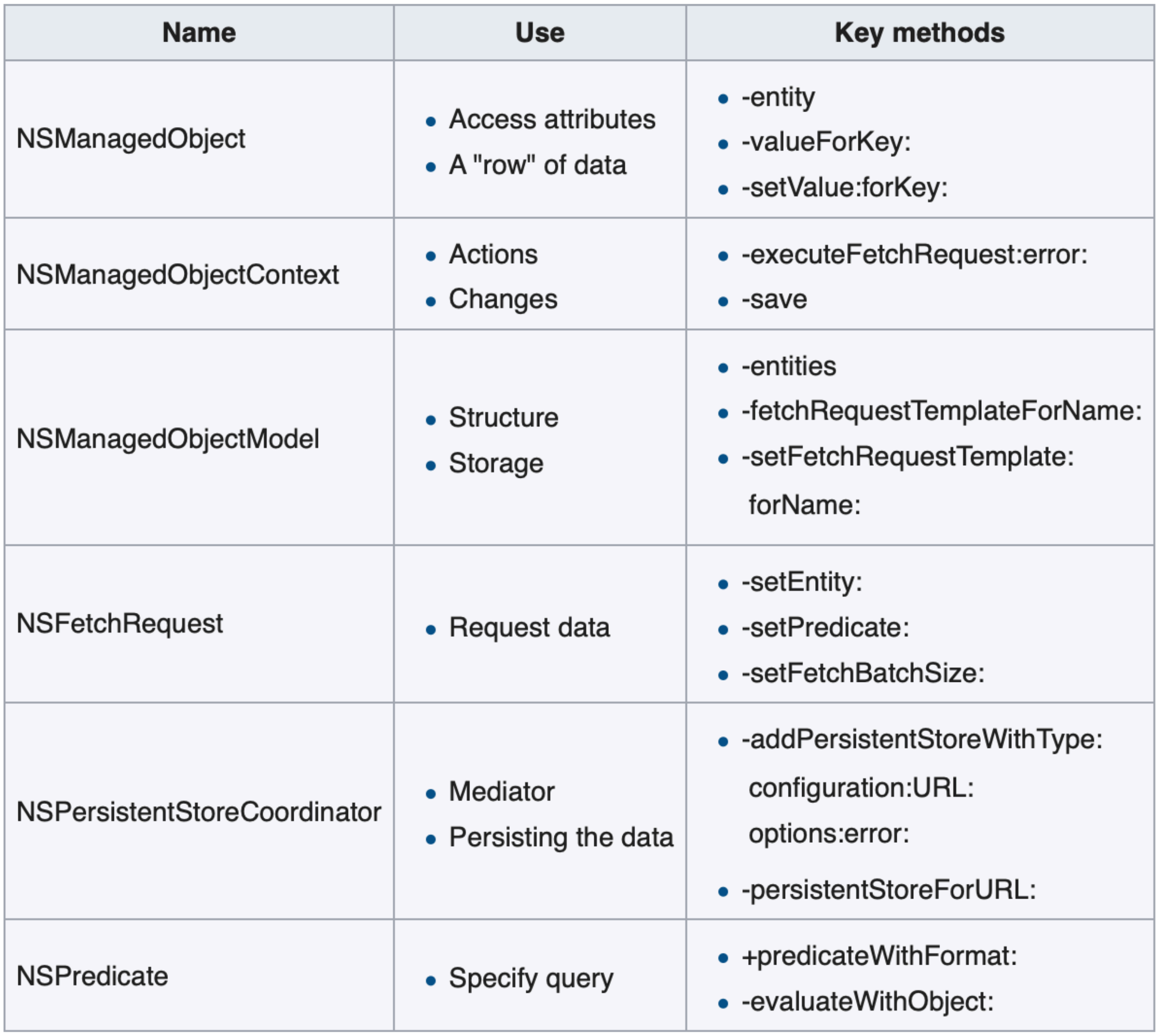
- An ARTIST has MANY ALBUMS
- An ARTIST has MANY SONGS
- An ALBUM belongs to ONE ARTIST
- An ALBUM has MANY SONGS
- A SONG belongs to ONE ALBUM → this case is complicated, because a song can be in different albums. But for this project I decided to create this as to-ONE relationship.
- A SONG belongs to ONE ARTIST → This case, as the one above, also can be open to interpretation, when it can be argued that a song might belong to more than one artist, but in this application I decided to build a to-ONE relationship.
- A PLAYLIST has MANY SONGS
- A SONG can be in MANY PLAYLISTS

After the Entities and Attributes are created in the editor, we have the option to visualize it in a graphic way.



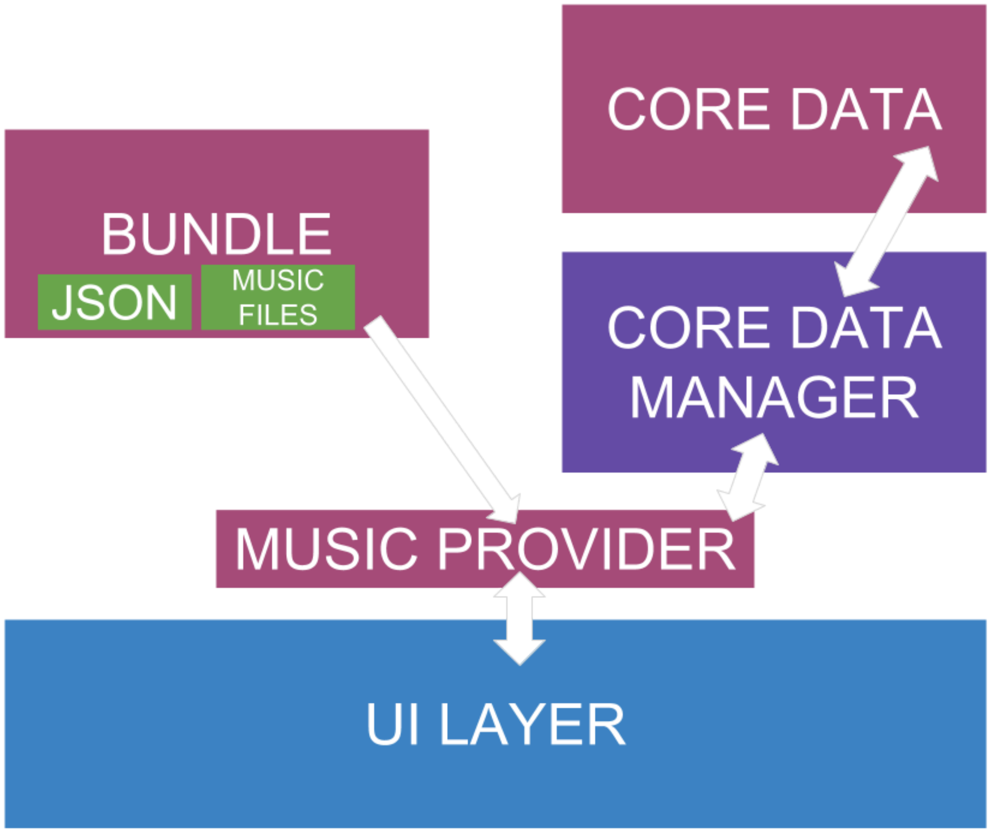
Every time a new Entity is created, xcode creates a Class for that entity. Each class is automatically create as a subclass of NSManagedObject.

NSManagedObjectContext is where all the NSManagedObjects are located.



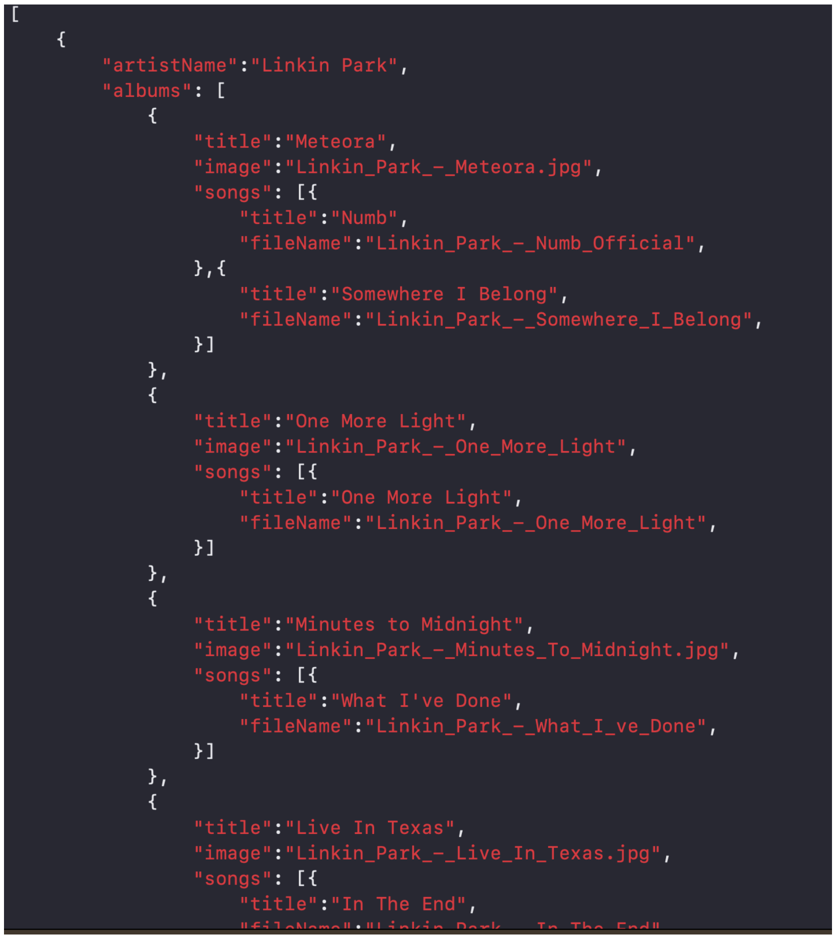
Name	Use	Key methods
NSManagedObject	• Access attributes • A "row" of data	• -entity • -valueForKey: • -setValue:forKey:
NSManagedObjectContext	• Actions • Changes	• -executeFetchRequest:error: • -save
NSManagedObjectModel	• Structure • Storage	• -entities • -fetchRequestTemplateForName: • -setFetchRequestTemplate: forName:
NSFetchRequest	• Request data	• -setEntity: • -setPredicate: • -setFetchBatchSize:
NSPersistentStoreCoordinator	• Mediator • Persisting the data	• -addPersistentStoreWithType: configuration:URL: options:error: • -persistentStoreForURL:
NSPredicate	• Specify query	• +predicateWithFormat: • -evaluateWithObject:

Behind the scenes



The Bundle

This layer contains the json file with all the data corresponding to the music files: Artists information, album titles, songs, etc, and the actual song files.



```
[
    {
        "artistName":"Linkin Park",
        "albums": [
            {
                "title":"Meteora",
                "image":"Linkin_Park_-_Meteora.jpg",
                "songs": [{
                    "title":"Numb",
                    "fileName":"Linkin_Park_-_Numb_Official",
                },{
                    "title":"Somewhere I Belong",
                    "fileName":"Linkin_Park_-_Somewhere_I_Belong",
                }]
            },
            {
                "title":"One More Light",
                "image":"Linkin_Park_-_One_More_Light",
                "songs": [{
                    "title":"One More Light",
                    "fileName":"Linkin_Park_-_One_More_Light",
                }]
            },
            {
                "title":"Minutes to Midnight",
                "image":"Linkin_Park_-_Minutes_To_Midnight.jpg",
                "songs": [{
                    "title":"What I've Done",
                    "fileName":"Linkin_Park_-_What_I_ve_Done",
                }]
            },
            {
                "title":"Live In Texas",
                "image":"Linkin_Park_-_Live_In_Texas.jpg",
                "songs": [{
                    "title":"In The End",
                    "fileName":"Linkin_Park_-_In_The_End"
```

The Music Provider

This layer is in charge of parsing the Json file with the music information and turning it into some readable data that my app can use to populate the song list.

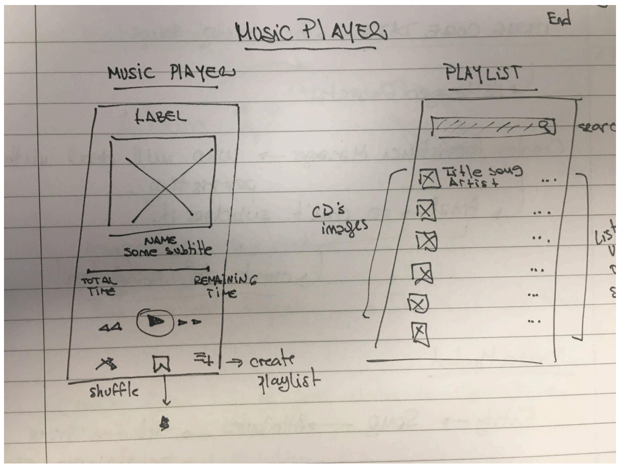
The Core data Manager

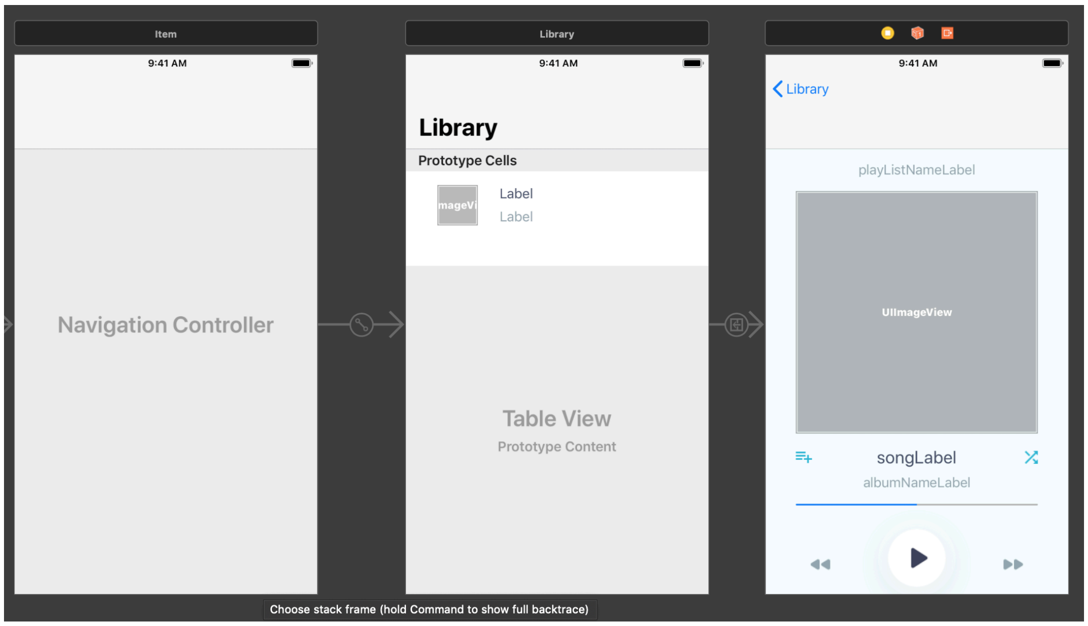
This Manager is a layer between Core data and the music provider. In this layer there are all the functions needed to write to the database and to retrieve information from the database.

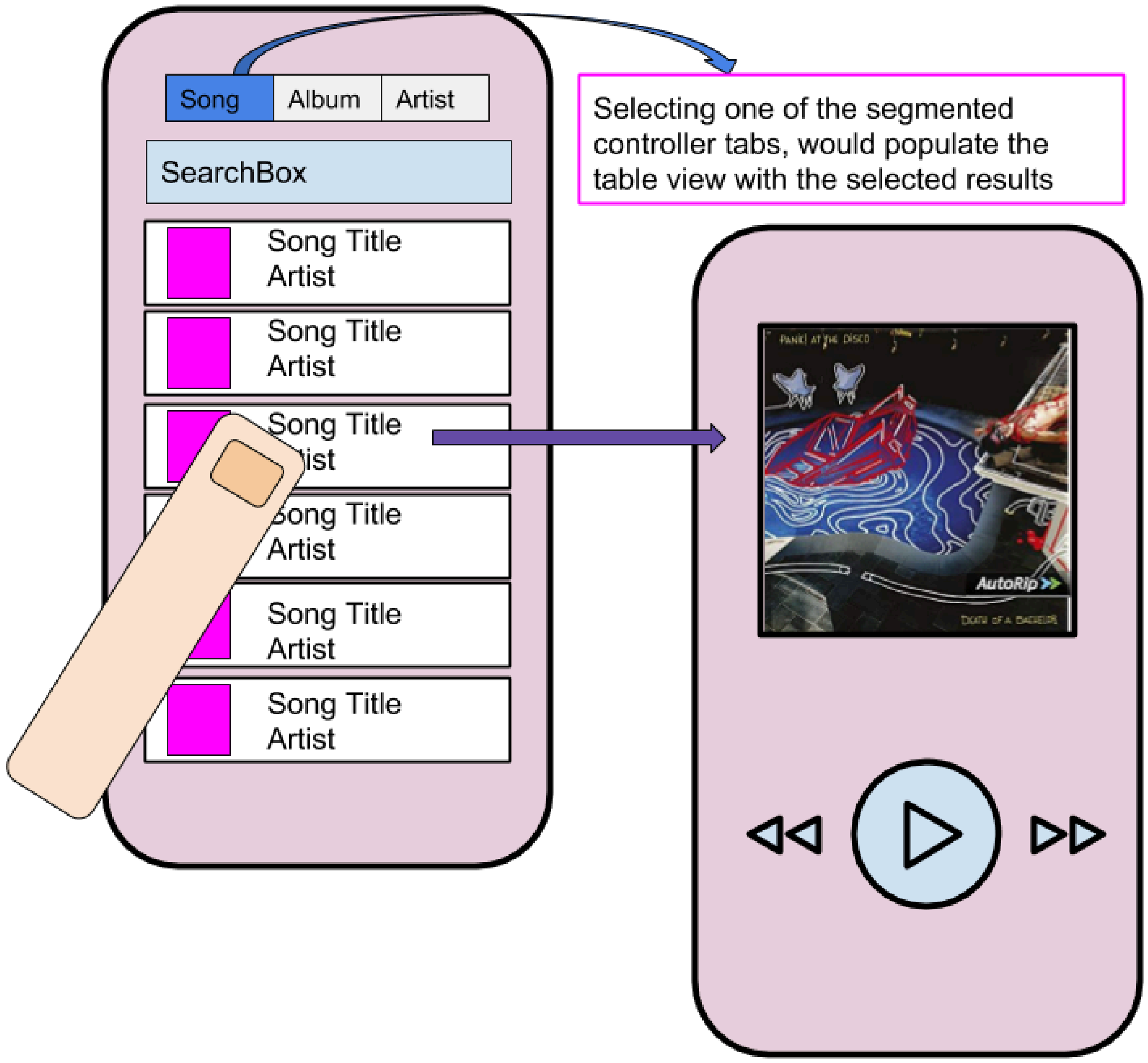
The UI Layer

This layer is in charge of displaying the data that the user can see on the app.

The UI design

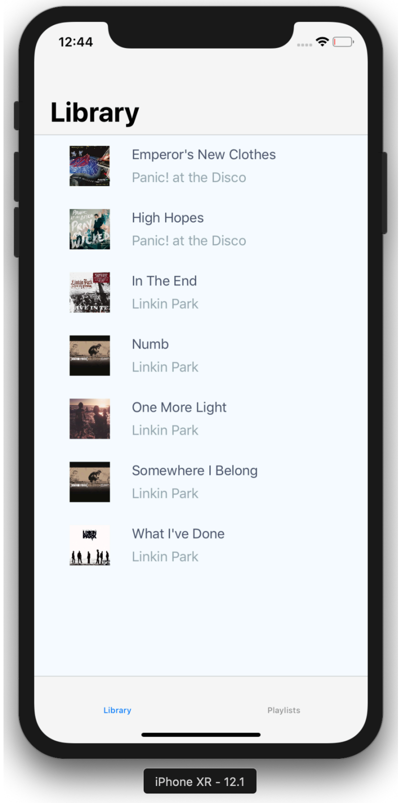





Song
Album
Artist
SearchBox
Song Title
Artist
Song Title
Artist
Song Title
Artist
Song Title
Artist
Song Title
Artist
Song Title
Artist
Selecting one of the segmented controller tabs, would populate the table view with the selected results
PANIC! AT THE DISCO
AutoRip
DEATH OF A BACHELOR

The music list view

This screen displays a list of music, that can be displayed by name of the song. Initially the design was to have many tabs where you could see the music by artist or album as well, but I then decided this wasn't necessary for an MVP. This screen also has a search bar that allows the user to filter the displayed results.



Emperor's New Clothes
Panic! at the Disco

High Hopes
Panic! at the Disco

In The End
Linkin Park

Numb
Linkin Park

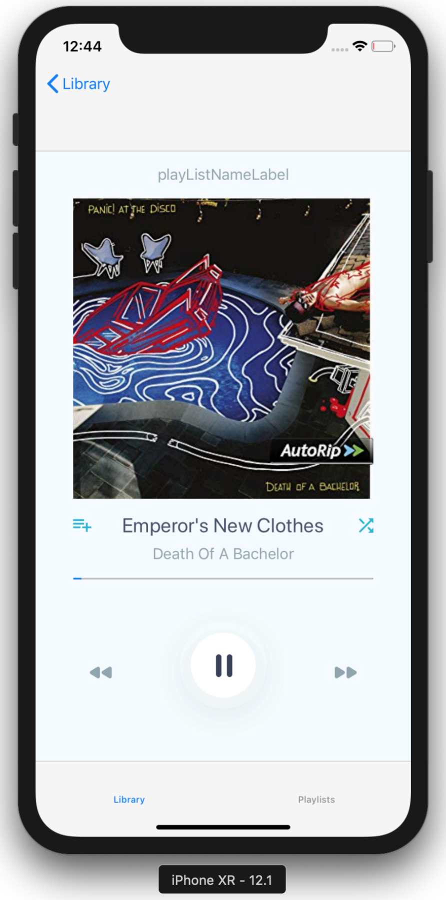
One More Light
Linkin Park

Somewhere I Belong
Linkin Park

What I've Done
Linkin Park

The music Player

This screen is the music player. When a row from the music list is selected, the screen transitions to this one, where you have the controls to play, pause, select next or previous song. You also have the options to add to a playlist (although this function is not implemented in this MVP) and to shuffle the music.



Assumptions and Limitations

As a personal challenge I decided to create this application using Swift, which is a language I used very little before. I decided to use this language because I was very interested in creating a native iOS app from scratch. The IDE is one of the most friendly I've ever used, and that encouraged me to take this challenge.

The first and biggest assumption I made was thinking that I would be able to create a Music Player, using Swift 4, in the timeframe given. This ended up being extremely challenging, and at the end I had to give up some features I would have loved to add.

Since my daily work does not involve creating or using databases, I think this was the most challenging part of the whole project, and assuming this would take one at the most was a wrong assumption.

The biggest limitation was the time: I would have liked to spend more time creating the UI, but because of the constraints, I had to go with a very simple, minimalistic design.

Although the app works, and is a decent MVP, there is still a lot of work to be done: fix many bugs!

Further Development

Improvements

Apple Music API

There's always room for improvement. While working on this project I found that there is an API for Apple Music, provided by Apple. Using this API would have been so much better than creating a Core Data database. I would only need to make requests to the API, and this would return the necessary data for me to display. Also I wouldn't have needed to manually add the songs, or create a Json file to provide with the information. But because this MusicKit API was very new, there wasn't enough documentation, and implementing seemed difficult for someone who is not very familiar with iOS libraries, etc. So, one of the improvements would be to add MusicKit to this project. This would give the user real access to their Apple music library.

Testing

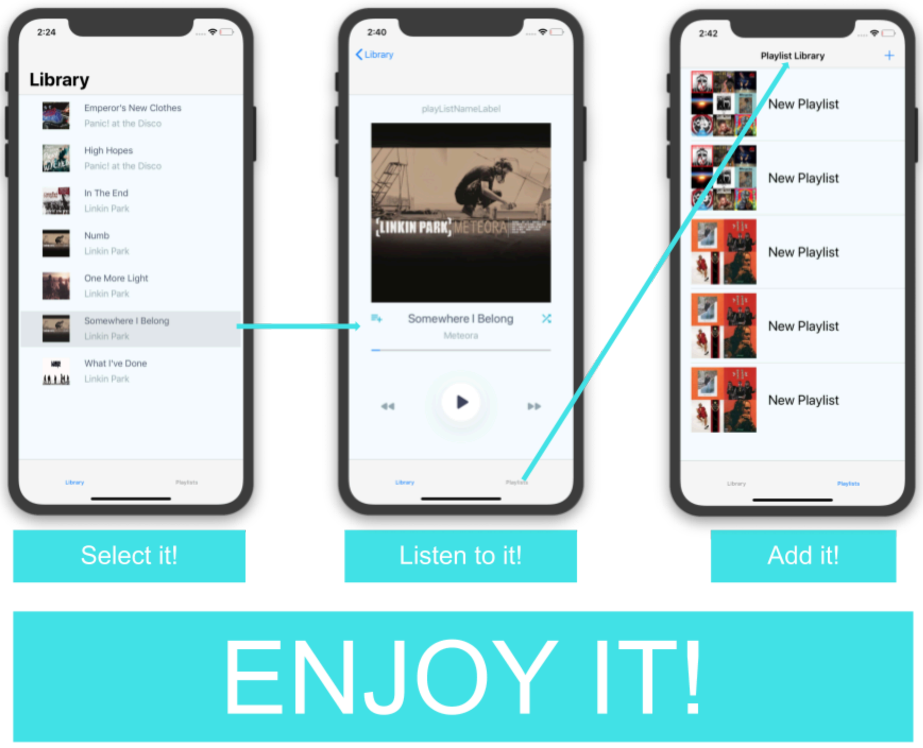
Because of the time constraints I couldn't make unit tests, which would have been the best approach for testing this app. Instead I manual tested it.

Fix the bugs!

Some of the bugs that would need future fixing:

- **The play/pause button** → when the app starts and plays the first song, it works perfectly, but as soon as I change to the next or previous song (or shuffle), the play/pause gets desynchronized and it could show a play button even if there is music playing already (where it should be showing a pause button).
- **Shuffle function:** This function picks randomly a song from the list of songs, but sometimes happens that it can choose on a row the same song, or selects the same songs over and over, and this is something that we wouldn't want in a music player app.
- **Playlist:** We can create a new playlist, and it gets assigned the name "New Playlist", and assigned a random image from a list of images I added myself. Ideally the user would be able to change the name of the playlist (not supported right now) and the image would be created as a combination of images of the albums of songs from the list itself, or could have an option from the user to handpick an image.
- The "create playlist" icon in the music player screen it's not attached to the playlist screen.
- The label on top of the image in the music player screen should have the name of the Band/artist.

User Guide



This music player it's very easy to use! You will see a list of you songs, to play one you just need to select one. On the music player you can play/pause the music, play the next track or previous track, shuffle and even create your own playlist! How cool is that? You can add as many tracks as you like to your playlist. In your song's list, you can search by name your favourites to make it easier.